Comparing Genetic Algorithm and Guided Local Search Methods by Symmetric TSP Instances

Mehrdad Nojoumian
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
mnojoumi@cs.uwaterloo.ca

Divya K. Nair
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
dknair@cs.uwaterloo.ca

ABSTRACT
This paper aims at comparing Genetic Algorithm (GA) and Guided Local Search (GLS) methods so as to scrutinize their behaviors. Authors apply the GLS program with the Fast Local Search (FLS), developed at University of Essex [24], and implement a genetic algorithm with partially-mapped and order crossovers, reciprocal and inversion mutations, and rank and tournament selections in order to experiment with various Travelling Salesman Problems. The paper then ends up with two prominent conclusions regarding the performance of these meta-heuristic techniques over wide range of symmetric-TSP instances. First, the GLS-FLS strategy on the s-TSP instances yields the most promising performance in terms of the near-optimality and the mean CPU time. Second, the GA results are comparable to GLS-FLS outcomes on the same s-TSP instances. In the other word, the GA is able to generate near optimal solutions with some compromise in the CPU time.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic methods.*

General Terms
Experimentation

Keywords
Genetic Algorithm, Guided Local Search, Fast Local Search, Travelling Salesman Problem

1. INTRODUCTION
The Travelling Salesman Problem (TSP) has been a widely accepted combinatorial optimization problem, studied for exploring the effectiveness of optimization techniques focused in seeking near optimal solutions to NP-hard problems. There is an abundance of GA approaches and Local Search (LS) heuristic methods proposed for finding better-quality solutions for the TSP problem within an acceptable time limit. Although, GA and LS methods aim at achieving the same goal of yielding near optimal solutions for the hard optimization problems, we are interested to analyze their problem solving behaviors, by executing two classes of methods on the standard s-TSP instances. The main concern regarding local search is the escape from the local optimum, once it is caught in the local optimum, it becomes almost impossible to advance further to more optimal solutions. This issue is handled easily by an existing array of techniques called meta-heuristics which are built on top of local search methods. One of the simplest and the oldest meta-heuristic is the Repeated Local Search [1] where a few restarts are applied each time the search is trapped in the local optimum. Some of the other existing meta-heuristics are Simulated Annealing [21], Tabu Search [22], etc. Unlike local search methods, the Genetic Algorithm method generates a finite set of random solutions at first and iteratively works towards generating better solutions in each successive step depending on a defined fitness function, in accordance with its selection schemes. The Guided Local Search proposed by Voudouris [2] is also one of the most effective heuristic search strategies, which directs the local search to improving areas of search space.

The major objective of this paper is to evaluate two well-known meta-heuristic methods by challenging the travelling salesman problem with various numbers of cities. This issue has been in the center of attention by scientists and engineers because numerous of other problems can be mapped into the TSP. Our motivation is to specifically focus on GA and GLS approaches in order to compare them by the assessment of their performance on symmetric TSP instances. We are interested to scrutinize behaviors of these algorithms on the solution space through various instances taken from TSPLIB [25].

The paper is organized as follows: Section 2 presents formal definitions, concepts and conventions used for better understanding of the paper. Section 3 reviews some related works regarding meta-heuristic approaches used for solving TSP instances. Section 4 and section 5 illustrate the proposed genetic algorithm and the guided local search method respectively. Section 6 demonstrates our experimental results. Finally, section 7 addresses concluding remarks and future works.

2. BACKGROUND
In the following parts, we describe some basic background material regarding combinatorial optimization, TSP and local search.

2.1 Combinatorial Optimization Problem

A combinatorial optimization problem can be defined as assigning values to a set of decision variables such that a function on these variables (objective function) is minimized when subjected to the specified set of constraints [10]. Hard combinatorial problems like the travelling salesman problem are challenging to be solved and the solving time grows exponentially with the size of the problem. There is no existing algorithm which can solve the TSP problem with a polynomial complexity. In fact, it is a prominent illustration of a class of problems in computational complexity theory which are classified as NP-hard [11].

2.2 Travelling Salesman Problem

Given a set of cities, n, and the distances among the cities, the TSP problem is to find a minimum-length tour such that every city is visited exactly once and returns to the starting point [20]. The formal definition as taken from [20] is as follows: Given a directed graph, Graph = (V, A) where V is the vertex set: {1...n} and A is the arc set: {(i, j): i and j Є V}, a cost factor: $C_{ij} \geq 0$ is associated with every arc. The TSP problem finds a partial digraph, (V, A_1) of G such that $|A_1| = n$ and for every vertex pair, v_1, v_2 Є V, there exist paths from v_1 to v_2 and v_2 to v_1 in G, whose tour cost, $\Sigma(i,j)$ Є A_1: C_{ij} is a minimum. The TSP problem is said to be symmetric if for every two cities A and B, the distance from A to B is equal to the distance from B to A.

2.3 Local Search

Local Search [3] is the search process by which the objective function, f(x) is minimized in a subsequent set of steps where in each step, the current solution, x, is substituted by another solution, y in such a way that: c(x) < c(y), y Є N(x). The N(x) is the neighborhood function of x which consists of all the solutions that can be reached from x by a single move [3]. The result of a move is a new solution obtained after necessary cost function modifications. The local search algorithm starts with an arbitrary solution and succeeds in a sequence of steps until a local optimum is reached. This local optimum is a solution: x which is defined with respect to its neighborhood: N(x) such that: c(x) < c(y), for all y Є N(x).

The local search procedure relies on two major moving strategies to perform the search process. The *first is the best improvement strategy* (greedy) where the current solution is replaced with the solution which has the most improving cost function compared to the cost functions of all its neighbors. On the other hand, the *first improvement strategy* replaces the current solution with another immediate improving solution as soon as it is discovered. The complexity of a local search procedure is dependant on the size of the neighborhood and the time required to compute a move.

The neighborhood structures used by local search for the TSP problem is defined by 'k-opt' moves. In 2-opt TSP, a new neighboring solution is generated from the current solution by performing a move such that two edges are deleted from the current solution and reversing any of the resulting paths and then joining the tour. In this paper we use the LS procedure based on 2-opt moves for solving the TSP instances. Figure 1 demonstrates the 2-opt move for TSP.

In the next section, we review some related works with respect to the genetic algorithm and guided local search methods.

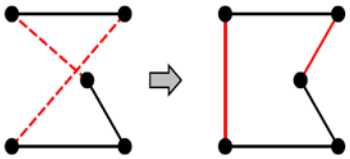

Figure 1. The 2-opt move.

3. RELATED WORK

There has been a plethora of significant works in the last two decades for solving travelling salesman problem, with the help of GA algorithm, heuristic search methods and a combination of the GA and heuristic search techniques, called hybrid methods.

Pullan [7] develops a combination of optimization heuristics and a phenotype (geometric) genetic algorithm for the TSP problem. In this approach, optimization techniques are used to locate the local optimum in the search space and the GA algorithm utilizes the local search space to identify the search location of the global optimum. The experimental results of this approach on various benchmark TSP instances show promising results in terms of mean CPU time and near optimal tour lengths for instances possessing less than 450 cities. The paper by Gang et al. [8] proposes a multiple heuristic search algorithm for the travelling salesman problem where two operators of complete 2-opt and Smallest Square (SS) are integrated to a genetic algorithm. The complete 2-opt is based on the traditional 2-opt heuristic and the SS strategy concentrates on shorter edges. The result of this approach reveals near optimal solutions for most of the TSP benchmark instances ranging from 100 to 1000 cities. Friesleben and Merz [9] illustrate a combination of local search heuristics and genetic algorithms for finding near optimal solutions in the travelling salesman problem. This approach ensures an improvement in the CPU execution time comparable to other LS and GA approaches. The experimental results confirm that executing a faster local search method frequently for a given time leads to a robust search algorithm that makes use of the search space efficiently with less CPU time.

The genetic approach by Ulder et al. [12] solves the TSP problem for eight standard benchmark TSP instances (city range: 48-666). This genetic algorithm is built on the order crossover and the Lin-Kernighan heuristic. The experimental results demonstrate the near optimum solution for each of this problem instance within 5 runs of the algorithm. The deviation from the known optimal solutions for these problems is found to be only 0.4% for all the tested eight TSP instances. Muhlenbein [16] introduces a parallel GA implementation for the TSP problem comprising of larger number of cities. In this approach, the population of chromosomes is very structured and the chromosomes are linked by a standard topology, thus easily conserving diversification in the population. The approach is guaranteed to derive the new local optimum which is better than the two local minima used for its generation. The computational results present near optimal solutions for large TSP instances like 442 and 532-city problems with a deviation of 0.6 from the known optimal solutions. The GA approach by [4] illustrates genetic operators such as crossover and mutation in solving the TSP problem using a flexible method. In this method, the generated solution is modified using genetic operators. The algorithm uses a verification strategy for the chromosome after the mutation to ensure that it improves the solution.

Zhang and Looks [5] present a new method for the traveling salesman problem which incorporates backbone information into the Lin-Kernighan (LK) local search family of algorithms for the problem. Their computational results on TSP instances scale up to 31623 cities. The new approach, called Backbone Guided Local Search (BGLS), outperforms LK by reducing the tour costs by more than 1% on average. Dong et al. [6] propose a novel tour construction heuristic for the TSP using the Less Flexibility First (LFF) principle. This new heuristic embraces the iterative improvement property similar to local search heuristics. The experimental results are comparable to the 3-opt and the simulated annealing. This approach outperforms other tour construction heuristics at the expense of increased running time.

As more researches were reviewed, we did not find any comparisons between the genetic algorithm and the guided local search methods. Therefore, we think this is a significant gap in the literature that needs to be filled because these two meta-heuristic approaches have some common behaviors.

In the next section, we illustrate the genetic algorithm and its implemented operations.

4. GENETIC ALGORITHM

The genetic algorithm refers to a model introduced and investigated by John Holland [18] and one of his students [19]. This algorithm encodes potential solutions to a specific problem on a chromosome-like data structure and applies recombination operators to these structures, such as cross over and mutation, in order to explore the solution space.

The Figure 2 represents the GA behaviour on the solution space of a potential problem. In the first step, it generates some random solutions, e.g. four solutions, such as $S_{1,0}$, $S_{2,0}$, $S_{3,0}$, and $S_{4,0}$. Afterwards, it continues investigating the solution space by the mutation operation. This operation usually helps the exploration of the solution space by going a few steps further from current positions, new solutions generated by mutation are presented by black circles. At the same time, the algorithm can apply another operation called cross over which combines various solutions to randomly breed new solutions, these results are presented by white circles. Finally, the algorithm selects some of these generated solutions based on a fitness evaluation in order to repeat the whole procedure for next iterations while keeping the best solution at each generation. The significant property of the GA is that it is able to search the entire solution space in a parallel way without getting stuck in local optimum. On the other hand, it may visit a set of solutions for several times, which is a time-consuming process.

As an example, Figure 3 presents a simple iteration of a GA on the travelling salesman problem. After generating first sample solutions in arrays, it applies GA operations. For the mutation, the algorithm just exchanges the position of two cities and for the cross over it fist splits each array into two parts and then combines counterparts together; the second part of the first array (7 2 4 6) with the fist part of the second array (5 1 3) and vice versa. In the next phase, the GA combines all new solutions and evaluates them based on the fitness function. In the TSP, this function returns the tour length of each solution. Finally, the algorithm selects some solutions (here solutions with shortest paths) as candidates of the new population for the subsequent iteration.

In the next discussions, the implemented mutation, cross over and selection strategies are illustrated.

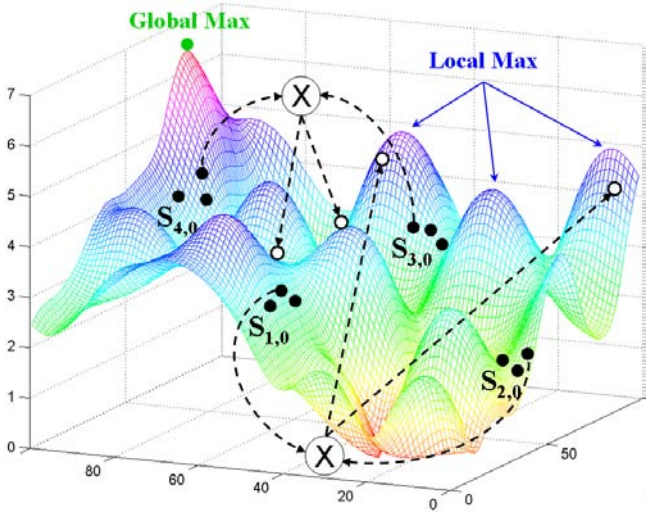

Figure 2. Genetic algorithm behavior on the solution space.

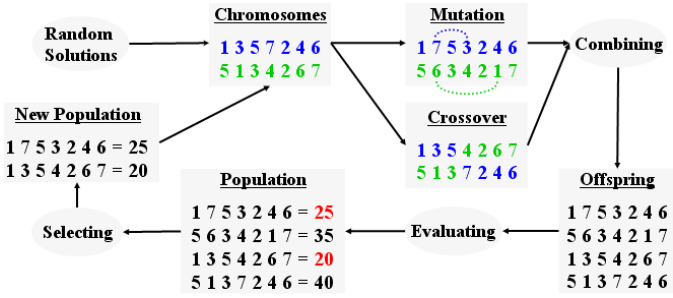

Figure 3. A genetic algorithm sample for the TSP.

4.1 Mutation

Two mutation methods are implemented for the TSP, Figure 4. In the first approach, called Inversion Mutation [13], we randomly choose two inversion points in the array and then invert the order of cities between two points. In the second technique, called Reciprocal Mutation [14], two cities are randomly chosen and then swapped.

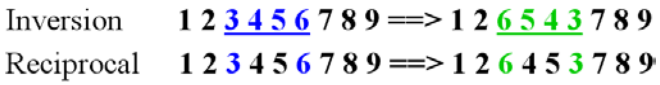

Figure 4. Mutation methods.

In our experiments, we got a better result regarding the inversion mutation because in the reciprocal method, the resulting solution is very close to the previous solution. Therefore, it can not greatly contribute to the optimization procedure, this issue is more critical for TSP instances with scores of cities.

4.2 Cross Over

We implemented two cross over strategies. In the first method, called *Partially-Mapped Cross Over* [15, 13, 17, and 14], we generate two crossover points randomly (A and B) and then copy the cities between A and B from the parent[1] into the child. Afterwards, for parts of the child's array outside the range [A, B], we copy only those cities from parent[2] which have not already

been taken from parent[1]. Finally, we fill in the gaps with cities that have not yet been taken, Figure 5.

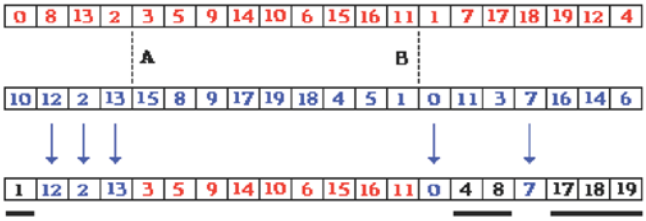

Figure 5. Partially-mapped cross over.

In the second approach, called *Order Cross Over* [17 and 14], we choose points A and B and copy that range from parent[1] to the child similar to the prior method. Subsequently, we fill in the remaining indexes with the unused cities in the order that they appear in parent[2], Figure 6. In the next part, two selection approaches are illustrated.

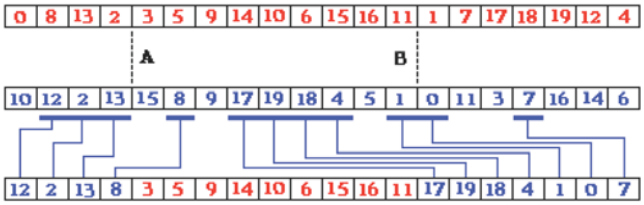

Figure 6. Order cross over.

4.3 Selection

The following two selection strategies are implemented: (a) *Rank Selection* and (b) *Tournament Selection* [14]. In the former method, we first sort the population by their fitness values and then assign a rank to each individual; rank one for the best solution, rank two for the second best solution and so on. After that, the probability of being selected is P^{Rank}. As an example, suppose P=0.5, in that case the probability distribution over individuals is as follow: $(0.5)^1 = 0.5$, $(0.5)^2 = 0.25$, $(0.5)^3 = 0.125$, etc. The P=1 is considered in our implementation.

In the second approach, we first pick a handful of N individuals from the population at random and then with a fixed probability P choose the one with the highest fitness value. Subsequently, we choose the second best individual with the probability $P*(1 - P)^1$, the third best individual with probability $P*((1 - P)^2)$ and so on. In our case, we consider P=1 and N=20 which means in each tournament only the best individual among N candidates is selected because the other probabilities are zero.

In both selection techniques, the tour length of each solution is considered as the fitness value. We got a better outcome with the rank selection since in the tournament strategy solutions with low quality have more chance to be chosen for the subsequent iterations while in the rank selection, we evaluate all solutions simultaneously, which means only high quality solutions have opportunities to go to the next iteration. In the next section, the guided local search method is illustrated.

5. Guided Local Search

In this paper, the s-TSP problem is evaluated using the GLS search procedure developed by Voudouris and Tsang [2]. The major property of the GLS is to escape from the local minimum resulting from the local search process and advance the search process further to promising regions of the search space. In the GLS, search information is converted into constraints on features which then are incorporated in the cost function using modifiable penalty terms. Constraints confine local search to the promising solutions with respect to the previous search information. GLS provides a simple mechanism for introducing or strengthening constraints on solutions features. Each time local search is trapped in a local minimum, the GLS can increment the penalty parameter of one or more of the features defined over solutions. If the penalty parameter of a feature is incremented, then solutions which have this feature are avoided by local search. A first step in the process of applying the GLS to a problem is to find a set of solution features that are responsible for part of the overall solution cost. Once features and their costs have been defined, GLS can be applied to the TSP.

The basic idea behind the GLS is devised as part of generalizing the GENET project [2] and incorporating the penalty concept from operations research [23]. The GLS improves the hill climbing behavior by introducing features on candidate solutions. A solution feature projects a specific property based on the major constraint expressed by the problem. For the s-TSP, a tour includes a number of edges and the solution cost (tour length) is given by the sum of the lengths of the edges in the tour. A set of features can be defined by considering all possible edges that may appear in a tour with feature costs given by edge lengths. For each feature, feature cost and penalty are defined. Feature cost for a feature depicts the effect of the solution properties on the solution cost and is derived from the objective function. For the s-TSP, the cost of a feature (edge included in the candidate tour from city A to city B) is the distance between A and B. A feature fi is expressed using an indicator function as:

$$I_i(s) \begin{cases} = 1, \text{solution s has property i}, s \in S, S = \text{set of all feasible solutions} \\ = 0, \text{otherwise} \end{cases}$$

In s-TSP problem, the indicator functions express the edges currently included in the candidate tour. As soon as the local minimum occurs during local search, the penalties are modified and the cost function is upgraded to a new augmented cost function based on the following equation:

$$h(s) = g(s) + \lambda . \sum_{i=1}^{M} P_i . I_i(s)$$

The g(s) is the objective function, M is the total set of features defined over solutions, λ is called the regularization parameter, p_i is the penalty associated with feature i. The regularization parameter decides the degree to which the solution cost is influenced by the penalties and guides the search process by recording this previous search information. Every time the local search encounters a local minimum, the penalty vector is defined as: $P = (P_1, \ldots, P_M)$ and the feature costs are defined by a cost vector: $C = (C_1, \ldots, C_M)$. On reaching a local minimum, the penalty parameters are increased by 1 in all features for which the following expression is the maximum:

$$\text{util}(S_*, f_i) = I_i(S_*) . \frac{Ci}{1 + Pi}$$

The penalty parameter keeps track of the number of times a particular feature is penalized. The term $C_i / (1+P_i)$ controls the utility function by ensuring that the value of this term decreases for a feature which is penalized for a number of times, and thus strengthening the diversification of search. The solution features

are penalized depending on the costs, that is, the features with higher costs are penalized more often than the features with lower costs. Since the penalty parameters of high cost features are increased, the solutions possessing these features will be neglected next time the local search procedure is called, and therefore search proceeds to more promising regions of the search space.

5.1 Guided Local Search Procedure

The GLS process is commenced by initializing all the penalty parameters to zero. At first, the local search procedure is called and local search proceeds until the first local minimum is reached. The first time and every other time a local minimum is encountered, the current cost function is modified to a new augmented cost function by incrementing the penalties of those features for which the utility function is a maximum. Then the local search procedure is invoked again by using the modified augmented cost function. Figure 7 presents the basic GLS idea; escaping from a local minimum in the landscape by increasing the objective function value of its solutions. Figure 8 describes the pseudo code for the guided local search process.

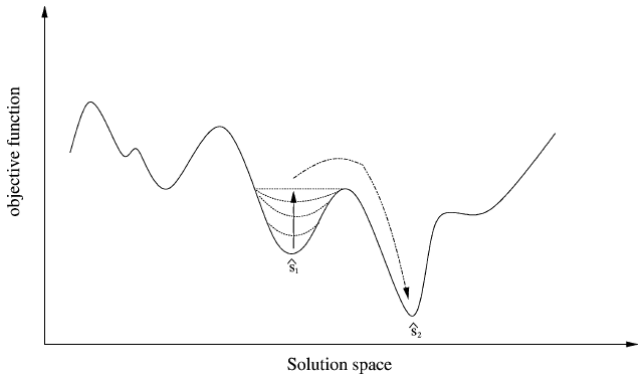

Figure 7. The basic GLS idea.

```
procedure GuidedLocalSeach(S, g, λ, [I1, ...,IM], [c1,...,cM], M)
begin
        k ← 0;
        s0 ← random or heuristically generated solution in S;
        for i ←1 until M do /* set all penalties to 0 */
                pi ← 0;
        while StoppingCriterion do
        begin
                h ← g + λ * ∑pi*Ii;
                sk+1 ← LocalSearch(sk, h);
                for i ←1 until M do
                        utili ← Ii(sk+1) * ci / (1+pi);
                for each i such that utili is maximum do
                        pi ← pi + 1;
                k ← k+1;
        end
        s* ← best solution found with respect to cost function g;
        return s*;
end
```

Figure 8. GSL pseudo code.

Where, S: search space, g: cost function, h: augmented cost function, λ: regularization parameter, I_i: indicator function for feature i, C_i: cost for feature i, M: number of features, P_i: penalty for feature i.

5.2 GLS with Fast Local Search (FLS)

One of the major improvements to the GLS process is the application of Fast Local Search (FLS). The FLS, by its nature, speeds up the search process since the problem space is disintegrated into a number of sub-neighborhoods and only a subset of these sub-neighborhoods are searched most of the time during search.

An activation bit is associated with each sub-neighborhood which will be turned on (equal to 1) only if it contains any improving moves, otherwise the bit is turned off (equal to 0). During each step of the search process, a specific number of sub-neighborhoods turn active depending on the current moves. As the search proceeds and reaches the final stage, all the sub-neighborhood bits become 0 since no further improving moves can be further detected. The guided fast local search starts by setting all the activation bits of sub-neighborhoods to zero. At first the FLS procedure is called and when the local minimum is reached, only the activation bits of the related sub-neighborhoods of penalized features are set to 1. Since only a subset of sub-neighborhoods is active at a particular time during the search procedure, the GLS process is accelerated to obtain the near optimal solution faster. The next section presents the experimental results of the paper.

6. EXPERIMENTAL RESULT

In the following section, comprehensive experimental results regarding Genetic Algorithm and Guided Local Search methods are demonstrated.

6.1 Comparison of GLS and GA on s-TSP

The results are reported for the symmetric TSP problem by extracting benchmark instances from the TSP Library, TSPLIB [25]. The s-TSP instances range from 48-1002 cities and covers wide range of distance permutations. The experiments are conducted using the IBM ThinkPad LENOVO, Intel Core 2 Duo CPU; 2.00 GHz with 2.00 GB of RAM and with the Windows XP as the operating system. We compared our results against the known optimal solutions for corresponding instances by a parameter, *Mean Excess %.* This parameter shows the deviation of our results from the known optimal solutions for the instances respectively and is calculated by: *Mean Excess% = [(Obtained Cost–Known Optimal Cost)/Known Optimal Cost] *100*

For GLS we used the GLS solver developed in C++ by [2]. This GLS Solver algorithm uses an N*N matrix for storing the distances between cities. It uses Fast Local Search and relies on the simple 2-Opt heuristic as the only move operator. If the regularization parameter (λ) is too large, then the selected moves removes the penalized features completely and sometimes may lead to wrong conclusions, but on the other hand, if the value is too small, then the search process will not be able to escape the local minima. Therefore, the regularization parameter, λ, is calculated by using the equation: $\lambda = \alpha$. [g (2-opt local min.)/ N], Where g (2-opt local min.) is the cost of 2-opt tour, N is the number of cities and the value of α lies between 0.125 and 0.5. For all the s-TSP instances, we choose the α value to be 0.125 since this value is best suited for 2-opt tour as reported in [3]. We

executed most of the s-TSP instances for about 10 runs with a time budget of 700 sec/run, and the number of iterations is hard coded in GLS solver to 200K.

The GA algorithm is implemented in Java. Most of the s-TSP instances were executed for 3-6 runs and the maximum number of iterations is set to 15000. For each s-TSP instance, we first generated (4*N, N-number of cities) number of random solutions and then applied the partially mapped crossover, the inverse mutation and the rank selection; since this combination yielded the best outcomes. Table 1 shows the experimental results of GLS-FLS and GLS-greedy LS and Figure 9 represents the graphical comparison between these two results. It can be deduced from the results that, for all the s-TSP instances both GLS-FLS and GLS-greedy LS compute optimal solutions. The performance of both GLS variants is almost the same regarding the optimal solutions, but the results show that GLS-greedy LS consumes more CPU time than GLS-FLS especially from *bier127* to *lin318*. This remarkable time difference exhibited by GLS-greedy LS is due to the inherent nature of the greedy-LS, since the whole neighborhood is searched to find the local minimum during each stage.

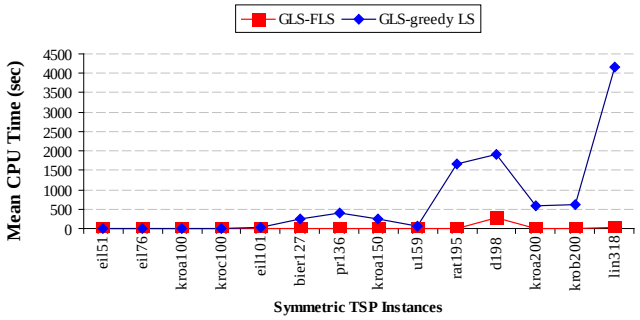

Figure 9. Graph-plot of GLS-FLS and GLS –greedy LS.

Table 1. Comparison of GLS-FLS and GLS –greedy LS.

TSP Instances	GLS-FLS			GLS- greedy LS		
	Mean CPU Time(Sec)	Number of Runs	Excess %	Mean CPU Time(Sec)	Number of Runs	Excess %
eil51	0.57	10	0	1.73	10	0
eil76	0.72	10	0	3.58	10	0
kroa100	1.7	10	0	11.72	10	0
kroc100	0.75	10	0	12.21	10	0
eil101	0.5	10	0	17.62	10	0
bier127	6.6	10	0	236.29	10	0
pr136	12.37	10	0	396.99	9	0.001
kroa150	7.55	10	0	257.58	10	0
u159	4.6	10	0	72.05	10	0
rat195	11.6	10	0	1656	8	0.01
d198	270	10	0	1914.4	10	0.08
kroa200	10.25	10	0	574.69	10	0
krob200	11.7	10	0	615.21	10	0
lin318	24.44	10	0	4162	8	0.01

Table 2 presents the experimental results of GLS-FLS-2-opt and GA on the selected s-TSP benchmark instances, Figure 10 demonstrates the graphical findings regarding the quality of solutions depicted by *Mean Excess %* and Figure 11 shows the graphical comparison of the two approaches in terms of *Mean CPU Time*.

The mentioned experimental results indicate that both approaches produce almost the same optimal solutions. This compelled us to deduce that the developed genetic algorithm and the GLS-FLS-2opt are comparable in terms of the near optimal solutions and generated almost the same results for most of the symmetric TSP instances. It is note-worthy that the GLS-FLS algorithm is set to 200K iterations while the GA algorithm is set to only 15K iterations. We also noticed that if the number of iteration in the genetic algorithm is increased then we can obtain even better solutions.

Although GA works very similar to GLS-FLS in terms of near optimal solutions, we observed a considerable time difference between the two approaches for the selected s-TSP instances. This difference is illustrated in Figure 11. This projected difference corresponds to the widely varying search behavior exhibited by both approaches. First of all, the GLS-FLS searches only a subset of sub-neighborhoods; moreover, the guided local search process, by its inherent nature, is guaranteed to proceed only to the promising regions of the search space. In contrast, the genetic algorithm process uses a random search behavior as it proceeds to distinct regions of the search space.

Table 2. Comparison of GLS-FLS-2opt and GA.

TSP Instances	GLS-FLS-2opt					GA				
	Mean CPU Time (Sec)	Tour Length	Number of Runs	Iteration	Excess %	Mean CPU Time (Sec)	Tour Length	Number of Runs	Iteration	Excess %
eil51	0.57	426	10	1307	0	3	438	6	2799	0.46
berlin52	0.26	7542	10	563	0	2.78	7542	6	1731	0
eil76	0.72	538	10	3141	0	13	557	6	4754	3.53
kroa100	1.7	21282	10	7485	0	25.2	21466	6	8942	0.86
kroc100	0.75	20749	10	9293	0	33.2	21096	6	6869	1.6
eil101	0.5	629	10	2315	0	27.9	651	6	4999	3.49
bier127	6.6	118282	10	32324	0	90.35	121089	6	9446	2.37
pr136	12.37	96772	10	42379	0	103.8	100986	6	14123	4.3
kroa150	7.55	26524	10	25902	0	225.3	28073	6	9100	5.8
u159	4.6	42080	10	20400	0	445.36	49811	6	12134	18.37
rat195	11.6	2323	10	32718	0	236.6	2467	6	14275	6.19
d198	270	15780	10	787244	0	264.1	16102	6	13369	2.04
kroa200	10.25	29368	10	39244	0	300.275	29368	6	14998	0
krob200	11.7	29437	10	24714	0	426.45	30122	6	14113	2.32
gil262	35.79	2378	10	95302	0	431.78	2452	6	6456	3.11
lin318	24.44	42029	10	151129	0	545.88	42029	6	14312	0
pcb442	298.65	50778	10	1954669	0	658	62556	6	15000	23.14
rat575	66.3	6776	3	272299	0.04	344.78	6776	3	5098	0.04
rat783	231.5	8809	3	959621	0.03	892.56	8915	3	15000	1.23
pr1002	279.9	259162	3	1304819	0.045	1008.67	260987	3	15000	0.74

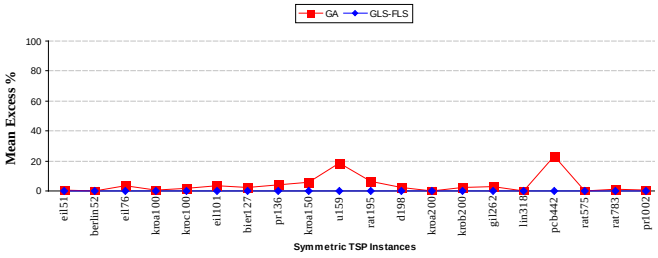

Figure 10. Graph-plot of GLS-FLS-2opt and GA on Mean Excess %.

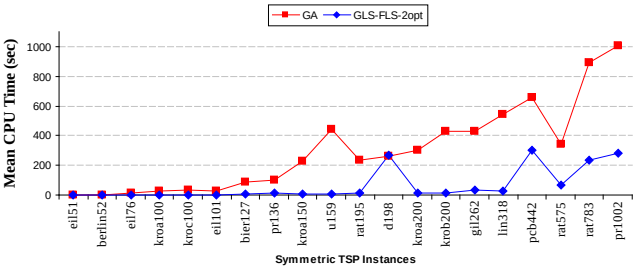

Figure 11. Graph-plot of GLS-FLS-2opt and GA on Mean CPU Time.

7. CONCLUSION AND FUTURE WORK

The paper first illustrates the genetic algorithm and guided local search methods and then demonstrates the implementation details of the GA and the experimental results regarding the mentioned algorithms. Authors then end up with two prominent conclusions regarding the performance of these meta-heuristic techniques over wide range of symmetric-TSP instances. First, the GLS-FLS strategy on the s-TSP instances yields the most promising performance in terms of the near-optimality and the mean CPU time. Second, the GA results are comparable to GLS-FLS outcomes on the same s-TSP instances. In the other word, the GA is able to generate near optimal solutions with some compromise in the CPU time.

Authors also observed that both the GLS-FLS and the GA take almost the same CPU time over multiple runs in order to solve the *d198*, which is different from other instances. Although, in most s-TSP instances cities are distributed all over the XY-coordinate or the density of cities on the XY-coordinate is almost around one region, the distribution of cities in the *d198* is dissimilar; multiple densities and multiple sparsities, Figure 12. As a future work, we intend to investigate this issue in details in order to see if we can come up with some new patterns in TSP instances. We also would like to execute the same s-TSP instances on the Genetic Local Search algorithm developed by [9]. Since this approach is based on the combination of the GLS and the GA; the GLS part of the approach would efficiently handle the local optimum and the GA part of the approach uses the search space of the local optimum to

ultimately find the global optimum, thus balancing intensification and diversification.

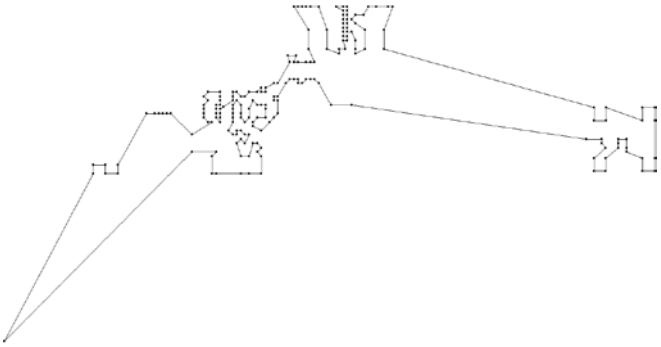

Figure 12. Distribution of cities in d198: drilling problem.

8. ACKNOWLEDGMENTS

We highly appreciate Professor Peter van Beek as well as the anonymous reviewers for their comments.

9. REFERENCES

[1] Voudouris, C. 2000. Guided Local Search Joins the Elite in Discrete Optimization. DIMACS Workshop on Constraint Programming and Large Scale Optimization, 29-40.

[2] Voudouris, C. and Tsang, E. 1998. Guided Local Search. European Journal of Operations Research 113, 80-119.

[3] Voudouris, C. 1997. Guided Local Search for Combinatorial Optimisation Problems, PhD Dissertation, Department of Computer Sciences, University of Essex.

[4] Karova, M., Smarkov, V., and Penev, S. 2005. Genetic Operators: Crossover and Mutation in Solving the TSP Problem. Conference on Computer Systems and Technologies, 6 pages.

[5] Zhang, W. and Looks, M. 2005. A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones, 19th International Joint Conference on Artificial Intelligence. 343-348.

[6] Dong, S., Guo, F., Yuan, J., Wang, R. and Hong, X. 2006. A Novel Tour Construction Heuristic for Traveling Salesman Problem Using LFF Principle. In Proceedings of the Joint Conf. on Information Sciences, Kaohsiung, Taiwan, 4 pages.

[7] Pullan, W. 2003. Adapting the Genetic Algorithm to the Travelling Salesman Problem. Evolutionary Computation 2, 1029-1035.

[8] Gang, P., Limura, L. and Nakayama, S. 2003. A Multiple Heuristic Search Algorithm for Solving Traveling Salesman Problem. Parallel and Distributed Computing, 779-783.

[9] Freisleben, B. and Merz, P. 1996. A Genetic Local Search Algorithm for Solving Symmetric TSP and Asymmetric TSP Problems, Conference on Evolutionary Computation, 616-621.

[10] Reeves, C. R. 1996. Modern Heuristic Techniques for Combinatorial Problems. John Wiley & Sons, Modern Heuristic Methods, 1-25.

[11] Reeves, C. R. and Beasely, J. E. 1993. Chapter 1: Modern Heuristic Techniques for Combinatorial Problems. John Wiley & Sons, Inc., 1-19.

[12] Ulder, N. L., Aarts, E. H., Bandelt, H., Laarhoven, P. J., and Pesch, E. 1991. Genetic Local Search Algorithms for the Travelling Salesman Problem. In Proceedings of the 1st Workshop on Parallel Problem Solving From Nature. Lecture Notes in Computer Science, vol. 496. Springer-Verlag, 109-116.

[13] Falkenauer, E. 1998. Genetic Algorithms and Grouping Problems. John Wiley and Sons.

[14] Michalewicz, Z. 1996. Genetic Algorithms + Data Structures = Evolution Programs (3rd Edition). Springer-Verlag New York.

[15] Jog, P., Suh, J. Y., and Gucht, D. V. 1991. Parallel Genetic Algorithms Applied to the Traveling Salesman Problem. SIAM Journal of Optimization, 1(4): 515-529.

[16] Muhlenbein, H. 1991. Evolution in Time and Space - The Parallel Genetic Algorithm. Foundations of Genetic Algorithms, 316-337.

[17] Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing.

[18] Holland, J. H. 1975. Adaptation in Natural and Artificial Systems, University of Michigan Press. Ann Arber.

[19] DeJong, K. 1975. An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD Dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

[20] Milano, M. and van Hoeve, W. J. 2002. Reducing Cost-based Ranking for Generating Promising Subproblems, In Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, Ithaca, USA, 1-16.

[21] Laarhoven, P. J. and Aarts, E. H. 1987. Simulated Annealing: Theory and Applications. Kluwer Academic Publishers.

[22] Laguna, M. and Glover, F. 1993. A Tabu Search Approach. Management Sci. 39, 492-500.

[23] Stone, L.D. 1983. The Process of Search Planning: Current Approaches and Continuing Problems. Operations Research 31, 207-233.

[24] The Guided Local Search Project: http://csp.bracil.net/gls.html

[25] Travelling Salesman Problem Library: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/